Bachelorarbeit

Zeitsynchrone Erfassung seismischer Sensordaten

Universität Bremen
Deutsches Zentrum für Luft- und Raumfahrt e.V.

Patrick Kleinermann
Matrikel-Nummer: 2589378

Betreuer	Norbert Toth
Erstprüfer	Prof. Dr.-Ing. Görschwin Fey
Zweitprüfer	Prof. Dr. Jan Peleska

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Symbolverzeichnis

Allgemeine Symbole

Symbol	Bedeutung
ADC	Analog-Digital Converter (Analog-Digital Wandler)
CCSDS	The Consultative Commitee for Space Data Systems
CPU	Central Processing Unit
GPS	Global Positioning System
NMEA	National Marine Electronics Association
PC	Program Counter
PPS	Pulse Per Second
PUS	Packet Utilization Standard
RA	Return Adress
ROBEX	Robotische Exploration unter Extrembedingungen

1 Einleitung

In diesem Abschnitt werden die Motivation zur Erstellung der Arbeit, die Ziele, der eigene Beitrag zur Verwirklichung der Ziele, sowie ein Überblick über die Struktur der Arbeit erläutert.

1.1 Motivation

Den Rahmen dieser Bachelorarbeit bildet das Projekt ROBEX[1]. Das Projekt ROBEX soll die Raumfahrt- und Tiefseeforschung zusammenführen. So sollen die Erfahrungen der Forschungsgebiete ausgetauscht werden und von beiderseitigem Nutzen sein. Das Hauptziel ist, die gleichen Methoden und technologischen Lösungen zu erarbeiten und im Anschluss in einer Demo-Mission im Jahr 2017 zu demonstrieren und zu bewerten [NAc].

Der Raumfahrtanteil des Projekts ist darauf ausgelegt die Mondoberfläche mittels seismischer Untersuchungen zu erforschen. Der ...

> Schwerpunkt dabei soll die Untersuchung der inneren Struktur sowie der Zusammensetzung der oberen Regolith-Schicht sein" [NAb].

1 Robotische Exploration unter Extrembedingungen

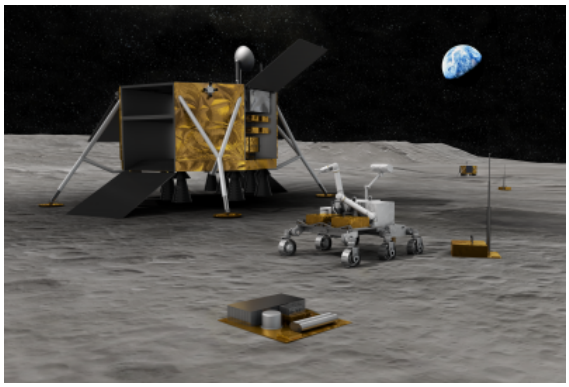

Abbildung 1.1: ROBEX-Visualisierung ©DLR-RY

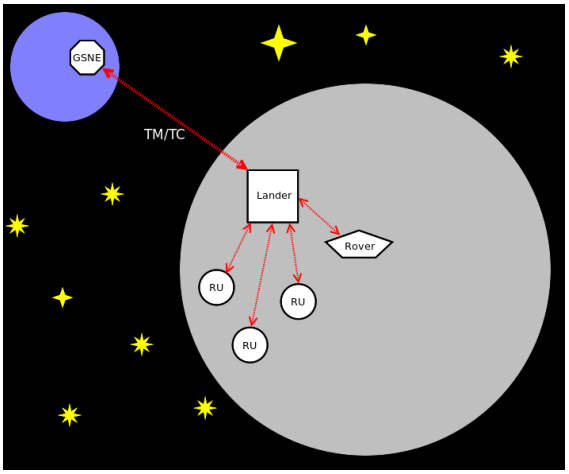

Abbildung 1.2: Kommunikation

In Abbildung 1.2 auf Seite 3 ist ein Überblick die Systemkomponenten für die Mondmission zu sehen. Es gibt auf der Erde ein *Ground Station Network Environment (GSNE)*, das Befehle an den, auf dem Mond befindlichen, *Lander* sendet und Daten von diesem empfängt. Das Ground Station Network Environment wertet die in den Experimenten gesammelten Daten aus. Der Lander dient als Dreh- und Angelpunkt für den Datenaustausch und die Energieversorgung der auf dem Mond befindlichen Komponenten.

Der Rover verteilt die Remote Units an verschiedenen Positionen. Diese Remote Units erfassen, unter Anderem, seismische Sensordaten und senden diese im Anschluss an den Lander. Der Lander wiederum sammelt diese Daten und schickt sie zur Auswertung an das *Ground Station Network Environment*. Die gesamte Kommunikation erfolgt hierbei über Drahtlose Verbindungen mit *Telecommands* und *Telemetry*.

Konkret befasst sich diese Bachelorarbeit mit den Remote Units. In diesen werden seismische Sensordaten erfasst, wobei verschiedene Abtastraten denkbar sind. So kann zu besonders interessanten Ereignissen die Abtastrate erhöht werden, um die Auswirkungen des Ereignisses genauer untersuchen zu können. In Ruhephasen hingegen können mit niedrigen Abtastraten die Ressourcen der messenden und auswertenden Systeme geschont werden.

Es gibt *aktive Experimente* und *passive Experimente*. Diese beiden Begrifflichkeiten bezeichnen zwei unterschiedliche Betriebsmodi eines Systems, z.B. einer Remote Unit. So ist ein *aktives Experiment* der Betriebsmodus, in welchem das System höhere Aktivität aufweist. Dies kann zum Beispiel die Erhöhung einer Datenerfassungsrate sein, oder auch die Aktivierung von Systembestandteilen, welche im *passiven Experiment* keine Funktion erfüllen. Das *passive Experiment* bezeichnet analog dazu den Betriebsmodus, in welchem geringere Aktivität vom System ausgeht.

Da Messungen von unterschiedlichen Remote Units stammen, kann die Varianz der Systemzeiten der Remote Units zur Uneinigkeit des realen Zeitpunkts führen, zu dem die Daten erfasst wurden. Um eine präzisere Auswertung solcher Daten zu ermöglichen, wird der Ansatz verfolgt, die Messungen zeitlich zu synchronisieren.

1.2 Ziele

Ziel dieser Bachelorarbeit ist es, mit einer Remote Unit seismische Sensordaten im *aktiven Experiment* mit einer Frequenz von 250Hz, sowie im *passiven Experiment* mit einer Frequenz von 50Hz zu erfassen. Sobald die Remote Unit aktiviert wird, startet automatisch ein *passives Experiment*. Ein *aktives Experiment* kann mittels Befehl initiiert werden. Ebenfalls kann mittels Befehl wieder das *passive Experiment* initiiert werden. Die Initiierung eines *aktiven Experiments* unterliegt derzeit noch keinen konkreten Bedingungen.

Die Daten dieser Experimente sollen mit der GPS-Zeit synchrnonisiert werden, damit eine Auswertung unabhängig von der On-Board-Zeit der konkreten Remote Unit möglich ist. Diese synchronisierten Daten gilt es in das *Extended Housekeeping Frame* einzubinden und an Lander zu senden. Dieser leitet die Daten an das Ground Station Network Environment, welches die Daten auswertet. Es gibt *Standard Housekeeping* Frames, die für Steuerdaten wie zum Beispiel die derzeitige Temperatur der CPU verwendet werden und es *Extended Housekeeping* Frames, mit denen Nutzdaten, wie Messungen oder Ähnliches übertragen werden. Die *Extended Housekeeping* Frames werden in regelmäßigen Zeitabständen von z.B. einer Sekunde übertragen.

Als Zeitquelle für die Synchronisation soll ein GPS-Empfänger verwendet werden, welcher mit einer Frequenz von 1 Hz die aktuelle GPS-Zeit empfangen und mittels Pulse Per Second (PPS)-Signal den Beginn einer Sekunde signalisieren kann. Die verwendete "epoche" ist der 01.01.2000 - 00:00:00 Uhr. Als Format für die Zeit soll CUC verwendet werden.

1.3 Eigener Beitrag

Der eigene Beitrag zu diesem Projekt ist die Implementierung einer Software, welche die Zeitsynchronisierung der Sensordaten realisiert, Befehle zur Steuerung des Betriebsablaufs auswertet und die Sensordaten versendet. Der Versand soll nicht direkt in das *Extended Housekeeping Frame* eingebunden werden, sondern stattdessen über die UART-Leitung in Form eines Comma-Separated-Value verschickt werden. Im Anschluss sollen die so Empfangenen Daten in einer Grafik ausgewertet werden.
Des Weiteren gehört zu dieser Arbeit die Wahl einer Synchronisierungsmethode, sowie eine zugehörige Genauigkeitsanalyse dieser Methode.

Die einzelnen Systemkomponenten sollen möglichst austauschbar und dementsprechend plattformunabhängig implementiert werden.

1.4 Überblick über das Dokument

Das erste Kapitel enthält die Einleitung. Hier wird der nähere Rahme erläutert, in dem diese Bachelorarbeit zustande kommt. Im Anschluss folgt im zweiten Kapitel eine Einführung in Grundlagen, wie zum Beispiel Zeitformaten, die zum Verständnis der Arbeit benötigt werden. Das dritte Kapitel beschreibt die einzelnen Komponenten des Systems und deren grundlegendes Zusammenspiel. Dort wird auch auf die konkret verwendete Hardware eingegangen. Das vierte Kapitel beschäftigt sich mit Protokollen, die nötig sind um mit einem GPS-Empfänger zu kommunizieren, als auch die Protokolle die dem Datenaustausch zwischen den Systemkomponenten. Das fünfte Kapitel erläutert die verwendete Synchronisationsmethode. Im sechsten Kapitel finden sich detailliertere Beschreibungen zu den internen Systemabläufen und der konkreten Programmierung. Das siebte Kapitel beschäftigt sich mit Tests, welche die Systemkomponenten und das Zusammenspiel

prüfen sollen. Das achte und letzte Kapitel bietet eine Zusammenfassung sowie einen Ausblick.

2 Grundlagen

In diesem Abschnitt werden die theoretischen und technischen Grundlagen erläutert, welche zum Verständnis für die konkrete Implementierung nötig sind. Hierunter fallen der Begriff der Zeitsynchronisierung, genutzte Zeitreferenzen, Zeitformate und Methoden für den Daten-Eingang sowie -Versand und die Aufnahme von Telecommands und Telemetrie-Daten.

2.1 Zeitsynchronisierung

Zeitsynchronisierung bezeichnet die Angleichung von mindestens zwei Zeitquellen. Sie kann auf verschieden Arten durchgeführt werden. So kann man eine der gegebenen Zeitquellen als Referenz festlegen und die anderen Zeitquellen an diese angleichen, indem die Zeit der Referenz übernommen wird. Eine weitere Möglichkeit ist es, die Abweichung der Zeitquellen voneinander festzustellen und als Versatz in die eigene Zeitdarstellung einzubinden.
Bei beiden Arten der Zeitsynchronisation muss diese mit hoher Wahrscheinlichkeit periodisch erfolgen, um die Abweichung der Synchronität über lange Zeit gering zu halten.
Die eben genannten Möglichkeiten sind in Abbildung 2.1 auf Seite 7 vereinfacht dargestellt. Die Zeitreferenz ist im linken und im rechten Beispiel rot.

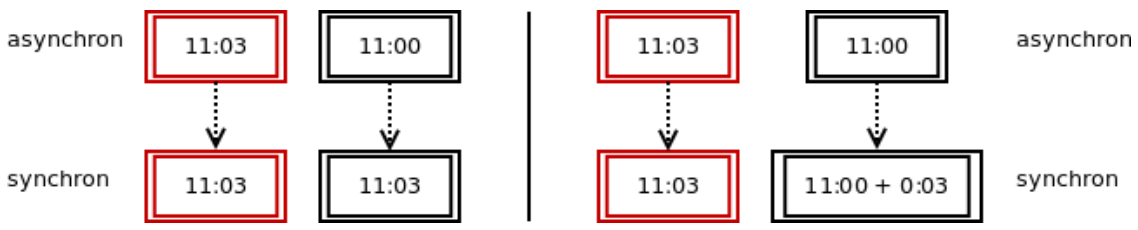

Abbildung 2.1: Zeitsynchronisation - Beispiele

2.2 Zeitreferenzen

Um eine Zeitsynchronisierung durchzuführen benötigt man eine Zeitreferenz, an welche man eine andere Zeitquelle angleichen möchte. In diesem Abschnitt werden zwei vorgestellt.

2.2.1 Global Positioning System

Das Global Positioning System (GPS) wird verwendet um sehr exakte Positionsermittlungen durchzuführen. Es stellt die sehr präzise GPS-Zeit zur Verfügung, auf welcher die Positionsbestimmung basiert. Der Dienst wird durch geoorbitale Satelliten erbracht, in denen sich jeweils vier Atomuhren befinden. Diese werden ihrerseits mit einer *master clock* synchronisiert, welche sich in den USA, Colorado befindet.
GPS kann als Zeitreferenz genutzt werden, sobald mindestens vier Satelliten sichtbar sind und vom Dienstentgegennehmer (GPS-Empfänger) lokalisiert wurden. So kann es sein, dass das Gerät bei sehr schlechter Wetterlage nicht benutzt werden kann. [KB06] [Pac96]

2.2.2 On-Board-Zeit

In der Regel haben alle Rechner eine Systemuhr, die den synchronisierten Ablauf des eigenen Systems koordiniert. Diese Systemuhr arbeitet mit einer Frequenz, die sich aus dem Zusammenspiel der Frequenz eines Schwingquartzes und einer Phasenregelschleife (Phase Locked Loop) ergibt. Im Weiteren wird die Systemuhr als Taktgeber bezeichnet.

2.2.2.1 Clock-Cycles und System-Ticks

Eine CPU wird als digitaler Schaltkreis mit Zuständen betrachten. diese Zustände werden durch einen Taktgeber in den jeweils nächsten Zustand übertragen. Die konkrete Anzahl dieser Zustände je Sekunde wird durch die Taktfrequenz angegeben. Anhand der Taktfrequenz ergibt sich die Dauer des kleinsten verfügbaren Zeitabschnitts. Diese ist $\frac{1}{Taktfrequenz}$ und ist die Dauer eines Clock-Cycles (CC).

Ein Clock-Cycle-Counter (CCC) zählt in der Regel die absolute Anzahl der bisher ausgeführten Clock-Cycles. Nach dem erreichen des Maximalwerts, beginnt der Clock-Cycle-Counter wieder bei 0. Den Clock-Cycle-Counter kann man als Zeitreferenz verwenden, für Synchronizationen, deren Zeitunterschied unterhalb des maximalen Wertebereichs des Clock-Cycle-Counters liegen. Die konkrete Dauer eines Clock-Cycles beträgt für die verwendete Hardware:

$$\text{CC-Dauer} = \frac{1\text{s}}{\text{Taktfrequenz}} = \frac{1\text{s}}{168\text{MHz}} = 5,952380\overline{9523}\,\text{ns} \tag{2.1}$$

Des Weiteren gibt es System-Ticks (ST). Diese System Ticks werden zu einer festdefinierten Anzahl von Clock-Cycles als Interrupt abgesetzt um einen Prozess/Thread-Wechsel zu ermöglichen. Da diese eine, um ein vielfaches geringere Frequenz aufweisen, sind sie nur als Zeitreferenz geeignet, sofern kein lesender Zugriff auf den Clock-Cycle-Counter möglich ist.

2.3 Zeitformate

Zeitformate dienen der wohldefinierten Darstellung von Zeit. So kann man Zeit als absolute Sekundanzahl ab einem definierten Zeitpunkt darstellen, aber auch andere Formatierungen sind denkbar. In diesem Abschnitt werden das Zeitformat der GPS-Zeit und der CCSDS Unsegmented Time Code vorgestellt.

2.3.1 GPS-Zeit

Die GPS-Zeit beschreibt die Anzahl von Sekunden ab dem Beginn des 06.01.1980. Sie weicht derzeit 17 Sekunden von der UTC-Zeit ab, da die GPS-Zeit keine Leap-Seconds beachtet. Leap Seconds wurden eingeführt, um die UTC-Zeit an die Erdrotation anzupassen, sodass sie von der UTC-Zeit höchstens 0,9 Sekunden abweicht. Eine Leap Second wird in einem Intervall von mehreren Jahren in die bestehende Zeit eingefügt[TSD15].

2.3.2 CCSDS Unsegmented Time Code (CUC)

Das Consultative Committee for Space for Space Data Systems (CCSDS) stellt eine Reihe an Standards für die Kommunikation und Datenhaltung im Weltraum bereit. So empfiehlt das CCSDS in ihren Bluebooks besonders geeignete Standards. Einer dieser Standards nennt sich *Time Code Formats*, in dem unter Anderem der CCSDS Unsegmented Time Code (CUC) vorgestellt wird[The10].

Es wird in unsegmentierte und segmentierte Zeitkodierungen eingeteilt. Unsegmentierte kodierungen stellen einen absoluten Sekundenzähler ab einem festdefinierten Zeitpunkt dar, der sogenannten *epoch*. Segmentierte Zeitkodierungen können Formatierungen haben, die nicht sekundenbasiert sind. Die *epoch* kann frei gewählt oder die Standard-*epoch* sein (01.01.1958).

Eine Zeitkodierung aus einem ”preamble-field (P)” und einem ”time specification-field (T)” bestehen. Das ”preamble-field” definiert eindeutig die Optionen, Parameter und die Kodierungsstruktur des T-Fields. Jedes dieser Felder besteht aus einer ganzen Zahl an Oktetten. Bei der CUC ist das *preamble-field (P)* optional.

Hier betrachtet wird die unsegmentierte Kodierung CUC. Das CUC-Format besteht aus 1 bis 4 Oktetten CUC Coarse-Time und 0 bis 3 Oktetten CUC Fine-Time. Die CUC Coarse Time stellt die derzeitige absolute Anzahl der Sekunden ausgehend von einer Epoche dar. Die CUC Fine-Time stellt den Sekundenbruchteil dar. In der Abbildung 2.2 auf Seite 10 sieht man eine CUC Zeit, die den Wert 256 darstellt, also den 01.01.2000 - 00:04:16 Uhr, gemessen ab der *epoch* 01.01.2000 - 00:00:00 Uhr.

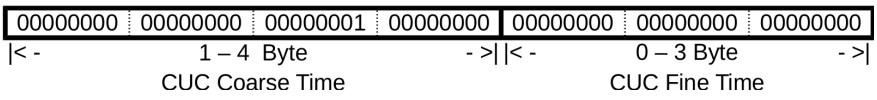

Abbildung 2.2: CUC Format Beispiel

Die kleinstmögliche Zeitquantisierung, die sich aus der Bitanzahl der CUC Fine Time ergibt ist:

$$\text{Kleinstmögliche Zeitquantisierung} = \frac{1\text{s}}{2^{|Bits|}} = \frac{1\text{s}}{2^{24}} \approx 59,6046447753906\text{ns} \tag{2.2}$$

2.4 Dateneingang

Erwartet man Daten von externen Geräten, so ist man darauf angewiesen mit diesen zu kommunizieren. Diese Kommunikation kann auf verschiedene Arten erfolgen. Eine Möglichkeit besteht darin, aktiv die Verfügbarkeit von Daten jedes externen Gerätes zu prüfen und die Daten daraufhin direkt abzugreifen (Polling). Je nach Beschaffenheit des externen Geräts kann Polling Sinnvoll sein, wenn zum Beispiel permanent aktuelle Daten verfügbar sind, die zu einem bestimmten Zeitpunkt abgegriffen werden sollen (Bsp: Spannungspegel). Sollten Daten zu unvorhersehbaren Zeitpunkten verfügbar sein, so ist diese Methode von Nachteil, da sie entweder sehr viel Rechenzeit beansprucht oder die Gefahr eines Datenverlustes (voller Datenpuffer) in Kauf genommen werden muss.
Eine gängige Möglichkeit solche unvorhersehbaren Dateneingänge zu behandeln ist es, z.B. nach jedem ausgeführten Befehl zu prüfen, ob ein externes Gerät die Verfügbarkeit von Daten signalisiert (Interrupt). Die genauere Definition der Prüfungszeitpunkte ist Implementierungs-spezifisch. Bei dieser Art Daten in Empfang zu nehmen ist es vorerst nicht wichtig, welches Gerät die Verfügbarkeit signalisiert hat. Ein Vorteil dieser Methode ist die ‚dass nicht unnötigerweise Rechenzeit aufgewandt werden muss, um die Anfrage auf Verfügbarkeit von Daten für jedes einzelne Gerät durchzuführen. In Abbildung 2.3 auf Seite 11 ist ein kleines Beispiel abgebildet, in dem nur ein Gerät Daten bereithält. Dieses Gerät ist grün markiert. Die einzelnen farbigen Linien stellen die sequenziell erfolgenden Anfragen auf die n Geräte dar. Beginnend bei 1 müssen n-2 Anfragen ausgeführt werden, deren Ergebnis negativ ist. Diese Anfragen haben also unnötige Rechenzeit beansprucht.

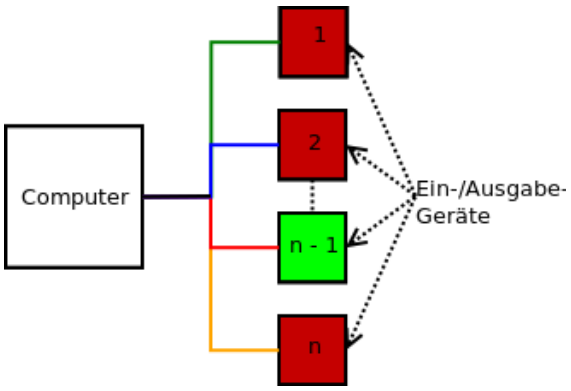

Abbildung 2.3: Polling

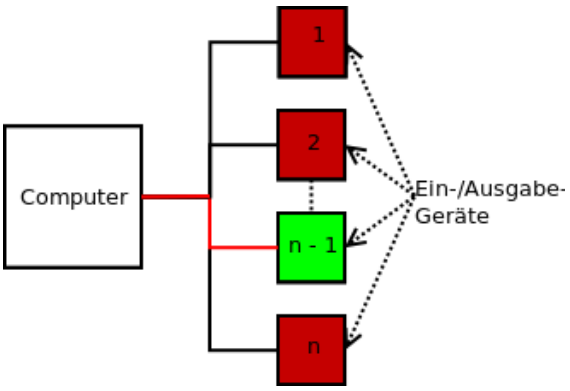

Abbildung 2.4: Interrupting

In Abbildung 2.4 auf Seite 11 ist ein ähnliches Beispiel abgebildet, in dem nur ein Gerät

Daten bereithält. Hierbei Signalisiert das Gerät n-1 dem Computer direkt, dass Daten verfügbar sind, die der Computer direkt abgreifen kann. Für diesen Fall wird keine unnötige Rechenzeit gebraucht.

2.4.1 Interrupting

Mittels eines Interrupt-Requests wird ein Interrupt-Flag aktiviert, welcher dem Prozessor die Verfügbarkeit von Daten signalisiert. Es existiert eine Interrupt Vector Table, in der die Geräte aufgelistet sind, welche einen Interrupt initiieren können (Seite 92,ff [Tan08]). Diese Listeneinträge sind direkt mit der Adresse der auszuführenden Interrupt Service Routine verknüpt. Eine Interrupt Service Routine ist eine Folge von Befehlen.
Interrupt Service Routinen können von anderen Interrupt Service Routinen unterbrochen werden. Daher kann jedem Interrupt-Request jeweils eine Zahl zugeordnet werden, welche eine Priorität angibt. Diese Priorität wird dazu genutzt eine Unterbrechung einer hochpriorisiertierten Interrupt Service Routine durch eine niedrigpriorisierte Interrupt Service Routine zu unterbinden.
Im umgekehrten Fall sorgt die Priorisierung dafür, explizit eine Unterbrechung zu erzwingen, sofern eine niedrigpriorisierte Interrupt Service Routine von einer hochpriorisierten Interrupt Service Routine unterbrochen werden soll. Falls während der Ausführung eines Interrupts, ein Interrupt höherer Priorität signalisiert wird, so wird der aktuelle Interrupt unterbrochen und der höher Priorisierte Interrupt wird ausgeführt. Nach Beendigung der Ausführung des höher priorisierten Interrupts wird der vorangegangene Interrupt weiter ausgeführt, bis dieser am Ende angelangt ist, oder er wieder vom einem höher priorisierten Interrupt unterbrochen wird.
Falls gewährleistet sein muss, dass ein Interrupt, auch falls dieser keine hohe Priorität hat, nicht unterbrochen werden soll, so besteht die Möglichkeit Interrupts zu deaktivieren. Hierfür werden möglichst früh zu Beginn der ISR die Interrupts deaktiviert und möglichst spät zum Ende wieder aktiviert. Hierbei ist zu beachten:

> If a processor takes too long to service a real-time clock interrupt or if it operates with interrupts disabled for more than one clock cycle, a clock interrupt will be missed and no error will be reported (Seite 236 [Com15]).

Es sollte also sichergestellt sein, dass die Interrupt Service Routinen schnell genug abgearbeitet werden, damit keine Ungenauigkeit der Systemzeit entsteht, weil ein Clock Tick nicht aufgenommen werden konnte.

2.4.2 Ablauf eines Interrupts

Als Ausgangssituation wird angenommen, dass ein laufender Prozess existiert, welcher in einem unprivilegierten Ausführungsmodus ausgeführt wird (Bps: User- Mode). Ein externes Gerät sendet einen Interrupt-Request, welcher ein Interrupt Flag setzt. Dieses Interrupt-Flag signalisiert dem Prozessor, dass Daten eines externen Geräts verfügbar sind.
Zu fest definierten Ereignissen, zum Beispiel der Ausführung einer Instruktion, wird vom Prozessor geprüft, ob ein Inferrupt-Flag gesetzt wurde. Bei dieser Prüfung wird entschieden, ob der Interrupt behandelt wird. Falls der Interrupt behandelt werden soll, so sichert der Prozessor den aktuellen Zustand und wechselt im Normalfall in einen privilegierten Ausführungsmodus (Bsp: Kernel Mode). Dieser Zustand besteht in der Regel aus dem aktuell vorliegenden Registersatz des Prozessors (insbesondere dem Program Counter (PC) und der Return Adress (RA)). Im Anschluss wird in der Interrupt Vector Table die Adresse der auszuführenden Interrupt Service Routine ausgewählt und in den Program Counter-Register geschrieben (Seite 340 [Tan08]). Danach folgt die Ausführung der Interrupt Service Routine.
Nach der Beendigung der Interrupt Service Routine wird der vorherige Zustand wieder hergestellt indem der zuvor gesicherte Registersatz wieder in die Register der CPU geschrieben wird und ein erneuter Wechsel in den vorherigen Ausführungsmodus geschieht. Am Ende wird die Ausführung an der vorherigen Stelle wieder fortgeführt. [Tan08] [Mur] [Sta12] [Com15]

3 Systemüberblick

In diesem Abschnitt wird näher erläutert, welche Hardware-Komponenten verwendet wurden und wie der Datenfluss aussieht.

3.1 Gesamtsystem

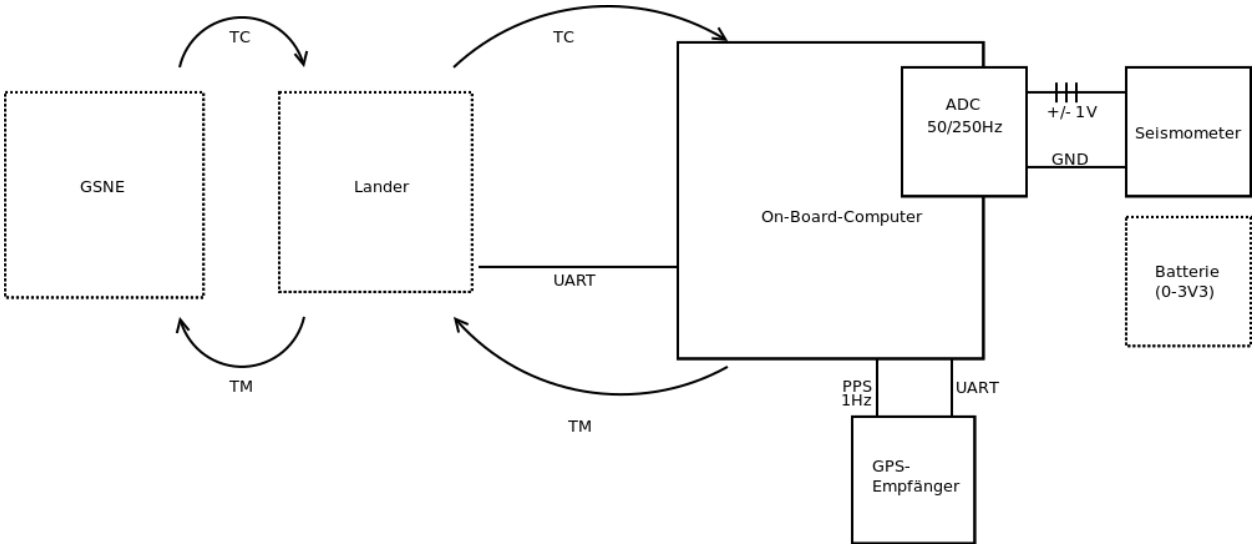

Abbildung 3.1: Allgemeiner Systemüberblick

3.1.1 Systemaufbau

In Abbildung 3.1 auf Seite 14 sieht man den allgemeinen Systemüberblick. Die dort sichtbaren Komponenten sind ein *On-Board-Computer* ein *ADC*, ein *Seismometer*, ein *GPS-Empfänger*, sowie auf der linken Seite der *Lander* und das *Ground Station Network Environment*. Es sei hier anzumerken, dass *Lander* und *Ground Station Network Environment* in der

konkreten Umsetzung durch einen Computer ersetzt werden. Deren Funktionen werden vom Benutzer ausgeführt.

Auf dem *On-Board-Computer* wird die *Zeitsynchrone Erfassung seismischer Sensordaten* durchgeführt. Der *ADC* dient dazu die analogen Signale des Seismometers in Digitale Signale zu Wandeln. Dies geschieht wahlweise mit einer Frequenz von 50 oder 250 Hz. Das Seismometer stellt auf drei Kanälen Daten für die x-, y- und z-Achse bereit. Die Daten liegen als Spannungsdifferenz zur Masse im Rahmen von -1V bis +1V an. Unter dem Seismometer ist eine Batterie angedeutet, welche im Entwicklungsprozess den Seismometer simuliert, da dieser noch nicht einsatzbereit ist. Der *GPS-Empfänger* liefert mit der Frequenz von 1 Hz jeweils ein PPS-Signal, sowie die benötigten Zeitdaten. Der *On-Board-Computer* konvertiert diese Daten, bereitet sie auf und sendet sie über Telemetry (via UART) an den *Lander*. Dieser leitet die Daten über Telemetry mittels Funkkommunikation an das *Ground Station Network Environment*, das die Daten auswertet. Das *Ground Station Network Environment* kann mittels Telecommands Befehle an den *Lander* senden, der diese wiederum dem *On-Board-Computer* schickt. So kann mittels Telecommands somit die Frequenz des ADC auf 50 oder 250Hz gesetzt werden.

3.2 Hardware

Im folgenden Abschnitt werden Hardware-Komponenten vorgestellt, die benötigt werden.

3.2.1 GPS-Empfänger

Um das Global Positioning System nutzen zu können ist es erforderlich eine Hardware zu verwenden, die diesen Dienst entgegennehmen kann. Ein GPS-Empfänger ist hierfür geeignet. Diese Geräte gibt es in verschiedenen Ausprägungen. Zum Beispiel, wie in dieser Arbeit verwendet, gibt es Geräte der Firma u-blox AG, deren Standard in Abschnitt 4.2 erwähnt ist. So gibt es Geräte, die auf dem TTL Level arbeiten und eine UART-Schnittstelle anbieten, mit denen man GPS-Daten auslesen kann. Zusätzlich zu den GPS-Daten kann es einen PPS-Output geben, der speziell auf hohe Geschwindigkeit ausgelegt ist und in

regelmäßigen Intervallen den Beginn einer Sekunde mittels einer Flanke signalisiert. Die Genauigkeit, die diese Geräte anbieten liegt bei ca. 10 Nanosekunden.

Bevor ein GPS-Receiver Daten empfangen und zur Verfügung stellen kann, muss ein Hot-Warm- oder Cold-Start erfolgen, der dafür sorgt, dass das Gerät betriebsbereit ist. Die verschiedenen Start-Arten benötigen unterschiedlich lange Zeit um die initialisierung zu beenden.
Bei einem Cold-Start werden alle im Gerät verfügbaren Daten gelöscht und neu ausgewertet. Hierbei handelt es sich um die *vorherige Position*, die Zeit, *Almanac-Daten* und *Ephemeris-Daten*.
Ein Cold-Start benötigt die längste Zeit. Diese Zeit ist wetterabhängig und beträgt ca. 35 Sekunden.
Bei einem Warm-Start werden nur die *Ephemeris-Daten* gelöscht und dementsprechend müssen nur diese Daten neu ausgewertet werden.
Ein Hotstart dauert nur wenige Sekunden. Hierbei sind alle vier zuvor erwähnten Daten vorhanden und müssen nicht neu ausgewertet werden. [SiRNA] [IncNA]

3.2.2 Pulse Per Second

Pulse per Second ist ein Signal, welches im Takt von, zum Beispiel 1 Hz, eine kurze Flanke sendet, um den Beginn einer Sekunde zu signalisieren. Diese Flanke kann konfiguriert werden in der Frequenz, sowie in der Dauer der Flanke. In Abbildung 3.2 auf Seite 17 entnommen aus [KB06], ist eine solche Flanke abgebildet. Man sieht dort als gestrichelte Linie eine Ideale Flanke und als stetige Linie eine wirklichkeitsgetreue Flanke. Dort sieht man im speziellen, dass aufgrund der Beschaffenheit der wirklichkeitsgetreuen Flanke eine minimale Ungenauigkeit entsteht. Da eine aufsteigende Flanke verwendet wird, um die Zeitsynchronisierung vorzunehmen sollte diese Tatsache der Ungenauigkeit in Betracht gezogen werden.

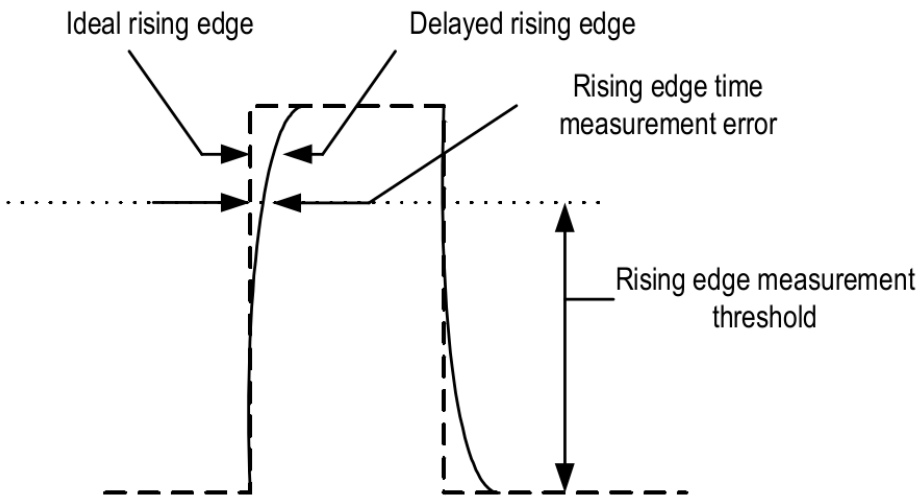

Abbildung 3.2: PPS-Flanke

Der verwendete GPS-Empfänger musste besondere Anforderungen erfüllen. Es musste auf dem TTLevel verfügbar sein, sowie über einen PPS-Ausgang verfügen.

3.2.2.1 Waveshare NEO-6M/7M GPS

Zum Empfangen der GPS-Zeit, sowie des PPS-Signals wird ein GPS-Empfänger der Firma Waveshare verwendet. Es handelt sich dabei um das Modell *NEO-6M/7M GPS* und ist ein *U-Blox*-Gerät, siehe Abbildung 3.3 auf Seite 17.

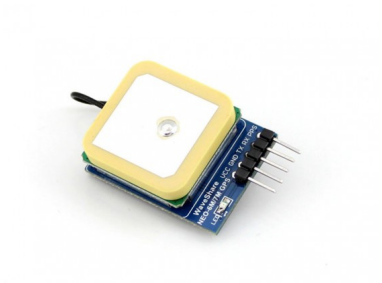

Abbildung 3.3: Waveshare NEO-6M/7M GPS
©Waveshare Electronics

Die Einbindung in das System erfolgt hardwareseitig mit eine Leitungsverbindung und nutzt UART. Über diese Verbindung werden die GPS-Daten übermittelt und Befehle

empfangen. Damit der GPS-Empfänger Daten empfängt ist es nötig ihm eine Spannungsversorgung (2,7V bis 5,0V) zur Verfügung zu stellen, da er keine eigene besitzt. Der GPS-Empfänger selbst besitzt eine Batterie, um Konfigurationsdaten für längere Zeit zu speichern. Somit ist es nicht nötig ihn ständig erneut zu konfigurieren. Das Gerät besitzt eine PPS-Leitung, über die das PPS-Signal übertragen wird.

Die Baudrate für den GPS-Receiver beträgt bei Standardeinstellung 9600bit/s. Das PPS-Signal sendet in der Standardkonfiguration jede Sekunde eine 100 ms andauernde Flanke.

U-Blox-Geräte haben zwei interne Zustände, die *current config* und die *permanent config*. Die permanent config wird bei Systemstart in die current config geladen. Jede Konfigurationseinstellung wird auf die current config angewendet. Falls eine Konfiguration langzeitig erhalten bleiben soll, muss sie mit einem Befehl in die permanent config übertragen werden.

3.2.3 Mikroprozessor

Der für diese Bachelorarbeit genutzte Mikroprozessor ist ein STM32F427ZGT6, welcher die Cortex-M4 Architektur implementiert. Im Mikroprozessor integriert befinden sich unter Anderem drei ADCs.

3.2.4 Analog-Digital-Wandler

In diesem Abschnitt werden die in dieser Arbeit verwendeten Analog-Digital-Wandler vorgestellt. Da das *Analog Devices AD7793EBZ Evaluation Module* noch nicht für diese Bachelorarbeit verwendet werden soll, sondern erst später in der endgültigen Umsetzung im Projekt, werden vorerst die internen Analog-Digital-Wandler des Mikroprozessors verwendet.

Analog-Digital-Wandler (ADC) dienen dazu analoge Signale in digitale Signale umzuwandeln. Sie besitzen eine definierte Bit-Breite, welche die Genauigkeit der konvertierten Signale beschreiben. Jede Konvertierung hat eine variierende Sampling-Time, die Abhängig von der Genauigkeit ist, die die Messung haben soll.

3.2.4.1 STM32F4 - Interne ADCs

Der Mikroprozessor enthält drei ADCs, die jeweils eine 12-Bit Quantisierung ermöglichen. Diese können welche synchron gestartet werden können. Nach dem Starten wird genau drei ADC-Clock-Cycles lang die Spannung des gewählten Kanals erfasst. Nach dem Ende der Erfassung folgt die tatsächliche Konvertierung der Daten, welche 12 ADC-CCs dauert, bei einer Quantisierung über 12-Bit. Die ADCs können keine Spannungen geringer als 0V aufnehmen. [Mic12]

3.2.4.2 Analog Devices AD7793EBZ Evaluation Module

Das Produkt der Firma Analog Devices unterstützt eine bis zu 24-Bit genaue Quantisierung und einen Spannungsbereich, der auch negativ sein kann.

3.2.4.3 Unterschiede der Hardware

Eine spätere Substituierung der Hardware wird nötig, da das endgültig zu verwendende Seismometer andere Spannungsbereiche als analoges Signal liefert, als die STM32-internen Analog-Digital-Wandler. Zudem kommt die gewünschte höhere Quantisierung der Messdaten, die 24-Bit betragen soll.

Die wichtigsten Unterschiede, die für die spätere Substituierung der internen Analog-Digital-Wandler durch das *Analog Devices AD7793EBZ Evaluation Module* von Bedeutung sind, sind Tabelle 3.1 auf Seite 19 sichtbar:

Eigenschaften	STM32-Intern	AD7793EBZ	Anforderungen ROBEX
Anzahl ADCs	3	1	1
Anzahl genutzter Kanäle	1 je ADC	3	3
Spannungsbereich	0V bis VREF(3V3)	-VREF bis +1VREF	-1V bis +1V
Quantisierung	12 Bit	24 Bit	24 Bit

Tabelle 3.1: Anforderungen an Analog-Digital-Wandler

In dieser Tabelle von größter Bedeutung ist die Möglichkeit des *Analog Devices AD7793EBZ Evaluation Module* negative Spannungsdifferenzen aufzunehmen, was den STM32-Internen Analog-Digital-Wandlern leider nicht möglich ist.

3.2.5 Seismiksensor / Analoge Signalquelle

Der Seismiksensor der Firme Lennartz kann mit dem jetzigen Hardware-Komponenten nicht verwendet werden, da es noch keine passende Schnittstelle für die Hardware gibt. Die Software soll jedoch darauf ausgelegt sein einen unkomplizierten Austausch zu ermöglichen. Aus diesem Grund sei hier nur erwähnt, dass zu einem späteren Zeitpunkt ein *Lennartz LE-xD* als Seismik-Sensor verwendet werden soll.
Bis dies möglich ist, werden die Analogen Signale durch Batterien und Frequenzgeneratoren simuliert. Hierbei ist bei einem späteren Austausch darauf zu achten, dass verschiedenartige Wertebereiche der Analog-Digital-Wandler vorliegen wie in Abschnitt 3.2.4.3 auf Seite 19 bereits beschrieben.

3.2.5.1 Lennartz LE-xD

Bei dem Seismiksensor handelt es sich um ein Produkt der Firma Lennartz. Das verwendete Modell gehört zur *Lennartz LE-xD*-Familie des Herstellers. Es besitzt drei Kanäle für X-, Y- und Z-Achse. Die Übertragung erfolgt hierbei über eine Analoge Spannung, welche in einem Bereich von -1V bis 1V für jeden Kanal liegt. Bis zur Einsatzbereitschaft benötigt ein solcher Seismiksensor zwischen 30 und 120 Sekunden *warm-up time* [ele11].

Auf Abbildung 3.4 auf Seite 21 ist ein solcher Seismiksensor abgebildet.

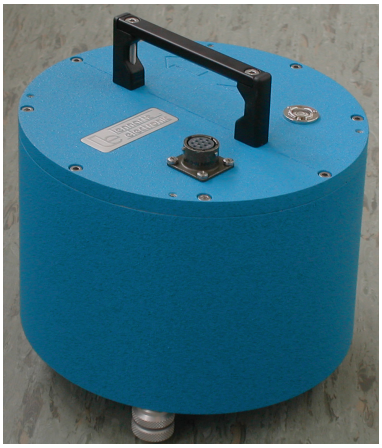

Abbildung 3.4: Produkt der Lennartz LE-xD-Familie ©

4 Protokolle

Protokolle spezifizieren Definitionen für einen Kommunikationsablauf und/oder Datenkapselung. Im Folgenden werden *NMEA 0183*, *U-Blox*, *Telemetry* und *Telecommand* vorgestellt.

4.1 NMEA 0183

Die National Marine Electronics Association hat ein NMEA 0183-Protokoll entwickelt. Das NMEA-Protokoll wird unter Anderem dazu verwendet GPS-Empfängern Daten zu übermitteln. Im speziellen kann ein GPS-Empfänger mit diesem Protokoll konfiguriert werden. Der Datentransfer funktioniert mit wohldefinierten NMEA Frames, in denen ASCII-Symbole verwendet werden. Die Definition findet sich unter Anderem in [AG08].

4.1.1 Aufbau des Frames

In Abbildung 4.1 auf Seite 23 sieht man den Aufbau eines NMEA-Frames. Es besteht aus einem *Start character*, einem *Address field*, einem oder mehreren *Data fields*, einer *Checksum*, sowie einer *End sequence*.
Der *Start character* ist immer das ASCII-Symbol '$' und kennzeichnet den Start eines Frames. Das *Address field* wird für die Unterscheidung von einzelnen *Standard Messages* verwendet. Das oder die *Data fields* beinhaltet die eigentlichen Nutzdaten. Hierbei ist insbesondere darauf zu achten, das der Beginn eines jeden neuen *Data field* mit einem Komma gekennzeichnet wird. Das *Checksum field* beinhaltet eine Prüfsumme, die es ermöglicht die Validität des empfangenen Frames zu beurteilen. Den Abschluss eines jeden Frames bildet die *End sequence*, welche immer aus den beiden Steuersymbolen <*CR*> (Carriage Return)

und <*LF*> (Line Feed) besteht. Ein Beispiel für ein korrektes NMEA Frame ist unten auf der Abbildung sichtbar.

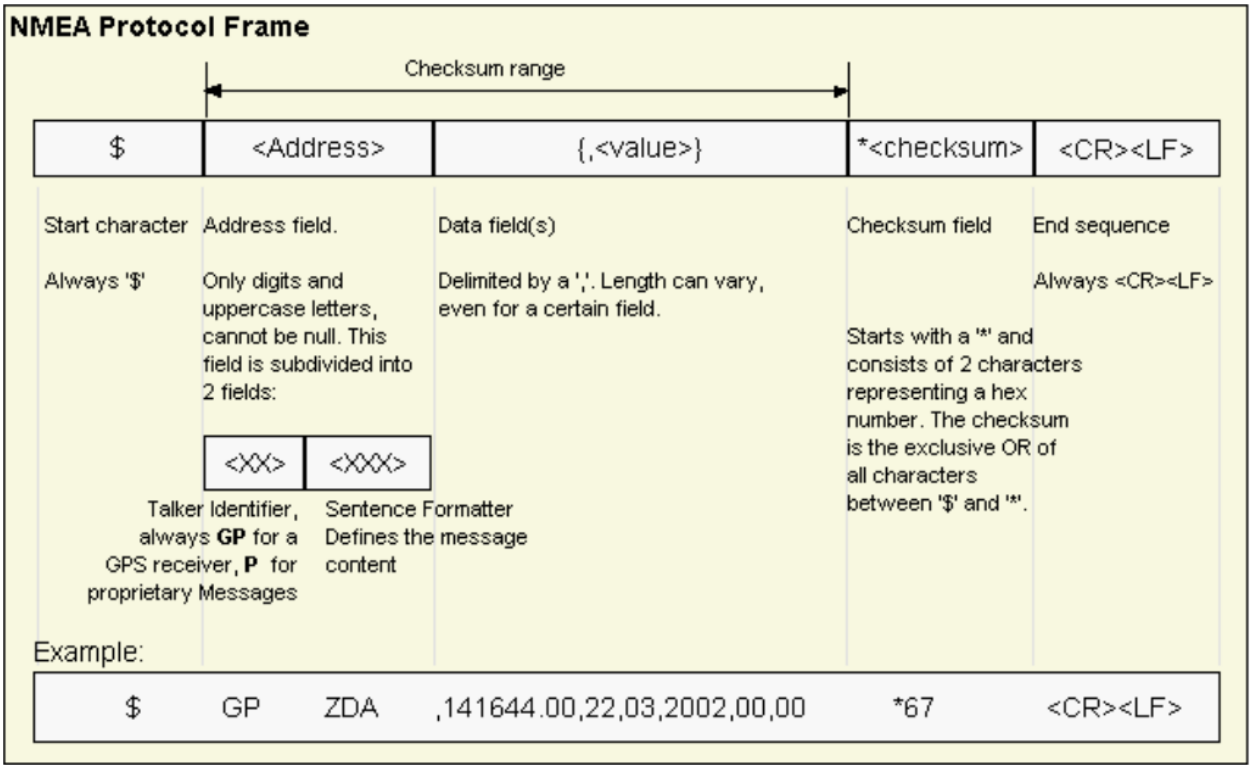

Abbildung 4.1: Allgemeiner Aufbau eines NMEA-Frames

4.1.2 Berechnung des Checksum Fields

Die Berechnung der Prüfsumme geschieht mit einer einfachen, byte-weisen XOR-Iterierung des *Adress field* und der *Data fields*. Die enthaltenen Kommas werden bei der Berechnung ignoriert.

4.1.3 Standard Messages

Es gibt 13 verschiedene *Standard Messages*. Diese *Standard Messages* sind jeweils auf einen bestimmten Zweck ausgerichtet, zum Beispiel Zeit- und Datumsangaben. Alle *Standard Messages* beginnen mit dem *Adress field*-Präfix 'GP'. In der Tabelle 4.1 sieht man eine kurze Beschreibung aller *Standard Messages* entnommen aus [AG08].

Adress field-suffix	Description
GGA	Global positioning system fix data
GLL	Latitude and longitude, with time of position fix and status
GSA	GPS Dilution of Precision and Active Satellites
GSV	GPS Satellites in View
RMC	Recommended Minimum data
VTG	Course over ground and Ground speed
GRS	Global Navigation Satellite System Range Residuals
GST	Global Navigation Satellite System Pseudo Range Error Statistics
ZDA	Time and Date
GBS	Global Navigation Satellite System Satellite Fault Detection
DTM	Datum Reference
GPQ	Poll Message
TXT	Text Transmission

Tabelle 4.1: NMEA-Standard Messages

First Data field	Number of Fields	Description
00	23	Lat/Long Position Data
03	5 + 6*GT	Satellite Status
04	12	Time of Day and Clock Information
04	4	Poll a PUBX message (ohne Daten)
40	11	Set NMEA message output rate
41	9	Set Protocols and Baudrate

Tabelle 4.2: NMEA-Proprietary Messages

4.1.4 Proprietary Messages

Neben den *Standard Messages* können auch sechs verschiedene Arten von *Proprietary Messages* benutzt werden. Diese Messages mit dem *Adress Field* 'PUBX' sind speziell für u-blox Geräte und dementsprechend von u-blox definiert. Es gibt sechs Arten solcher Messages, die in [AG08] beschrieben werden. In der Tabelle 4.2 sieht man eine kurze Beschreibung aller *Proprietary Messages* entnommen aus [AG08].

4.1.5 Verwendete Befehle

Für den Start und die Konfiguration des GPS-Empfängers wurden diverse NMEA-Befehle ausgeführt, welche im Folgenden erläutert werden.

4.1.5.1 De-/aktivierung von NMEA-Standard Messages

Im Rahmen dieser Bachelorarbeit sind nur der Daten der GPZDA-Standard Message von Nutzen. Aus diesem Grund wurden in der Vorbereitung alle nicht benötigten, und vorher aktiviereten, Standard Messages deaktiviert. Die GPZDA-Standard Message wurde explizit aktiviert. Dies geschah mit den folgenden Befehlen:

```
$PUBX,40,GGA,0,0,0,0*5A\n\r
$PUBX,40,GLL,0,0,0,0*5C\n\r
$PUBX,40,RMC,0,0,0,0*47\n\r
$PUBX,40,GSA,0,0,0,0*4E\n\r
$PUBX,40,GSV,0,0,0,0*59\n\r
$PUBX,40,VTG,0,0,0,0*5E\n\r
$PUBX,40,GRS,0,0,0,0*5D\n\r
$PUBX,40,GST,0,0,0,0*5B\n\r
$PUBX,40,ZDA,1,1,1,0*45\n\r
```

Die ersten acht Befehle deaktivieren die Ausgabe für DDC, USART1, USART2 und USB, jeweils durch eine 0 dargestellt. Der letzte Befehl aktiviert die Ausgabe für eben Genannte.

4.2 U-Blox

Das UBX Protokoll ist proprietär und dient dem Zweck GPS-Daten über eine asynchrone RS232-Schnittstelle zu senden. Das hierfür verwendete Paket-Format basiert auf einer Hexadezimal-Kodierung und ist nicht Menschenlesbar.

4.2.1 Aufbau des Pakets

In Abbildung 4.2 auf Seite 26 sieht man den Aufbau eines UBX-Pakets. Ein UBX-Paket besteht aus zwei *Sync Chars*, einer *Class*, einer *ID*, einer *Length*, einer *Payload*, sowie den Prüfsummen *CK_A* und *CK_B*.
Die zwei *Sync Chars* lassen den Anfang eines Pakets erkennen. Die *Class* bezeichnet die

Klasse der Messages und die *ID* den konkreten Message-Typ. *Length* gibt die Länge der Payloads in Byte an. Hierbei ist zu beachten, dass die Darstellung in *Little Endian* erfolgt, also das niederwertigste Byte auf der linken Seite steht. Dementsprechend kommt das niederwertigste Byte als erstes an. Die *Payload* enthält Informationen für den gewählten Befehl, zum Beispiel das Setzen bestimmter Werte in der Konfiguration eines GPS-Empfängers. Die beiden Prüfsummen *CK_A* und *CK_B* dienen dazu die Validität des empfangenen Pakets zu prüfen.

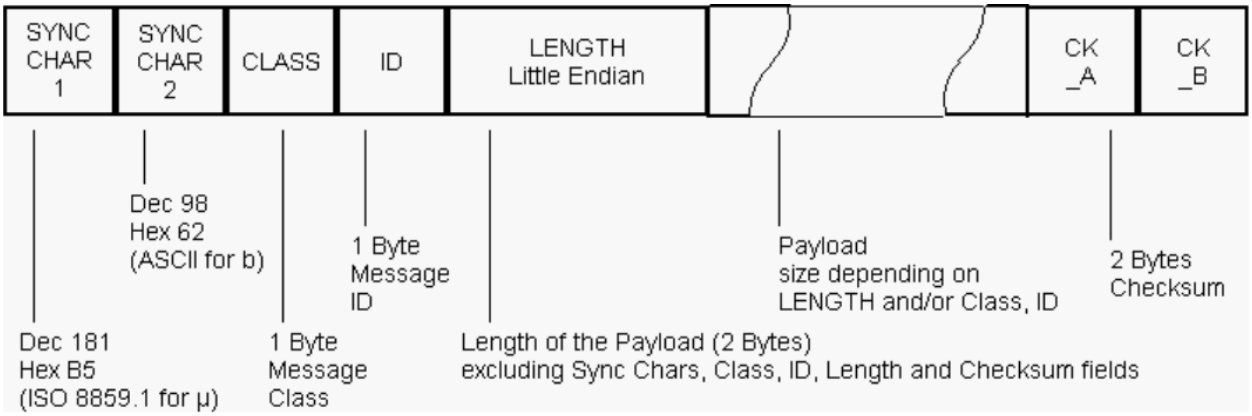

Abbildung 4.2: Allgemeiner Aufbau eines UBX-Pakets

4.2.2 Berechnung der Prüfsummen

Der verwendete Algorithmus, um die Prüfsummen zu berechnen ist der 8-Bit Fletcher Algorithmus, der im TCP Standard zum einsatz kommt. Der Bereich, über den die Prüfsummen gebildet werden erstreckt sich von *Class* bis einschließlich dem letzten Byte der *Payload*. Ein möglicher Quellcode, entnommen aus [AG08], sieht folgendermaßen aus:

```
CK_A = 0, CK_B = 0
For(I=0;I<N;I++)
{
  CK_A = CK_A + Buffer[I]
  CK_B = CK_B + CK_A
}
```

Der Algorithmus bildet für *CK_A* die Summe über jedes Byte des zuvor erwähnten Bereichs. Für *CK_B* wird in jedem Iterationsschritt um den, zu diesem Schritt aktuellen, Wert von *CK_A* erhöht.

4.2.3 UBX Class IDs

Es existieren acht unterschiedliche Class IDs. Diese werden mitsamt einer kurzen Beschreibung in Tabelle 4.3 auf Seite 27 dargestellt und sind entnommen aus [AG08].

Name	Class	Description
AID	0x01	Navigation Results: Position, Speed, Time, Acc, Heading, DOP, SVs used
RXM	0x02	Receiver Manager Messages: Satellite Status, RTC Status
INF	0x04	Information Messages: Printf-Style Messages, with IDs such as Error, Warning, Notice
ACK	0x05	Ack/Nack Messages: as replies to CFG Input Messages
CFG	0x06	Configuration Input Messages: Set Dynamic Model, Set DOP mask, Set Baud Rate, etc.
MON	0x0A	Monitoring Messages: Communication Status, CPU Load, Stack Usage, Task Status
AID	0x0B	AssistNow Aiding Messages: Ephemeris, Almanac, other A-GPS data input
TIM	0x0D	Timing Messages: Timepulse Output, Timemark Results

Tabelle 4.3: UBX Class IDs

4.2.4 Verwendete Befehle

Für die Konfiguration des GPS-Empfängers wurden diverse U-Blox-Befehle ausgeführt, welche im Folgenden erläutert werden.

4.2.4.1 Start des GPS-Empfängers

Zur Inbetriebnahme des GPS-Empfängers war es nötig diesen mittels eines Befehls zu Starten, nachdem die Spannungsversorgung für das Gerät aktiviert war. Die nötige Befehl lautet wie folgt:

```
Hot-Start: B5 62 06 04 04 00 00 00 02 00 10 68
Warm-Start: B5 62 06 04 04 00 01 00 02 00 11 6C
Cold-Start: B5 62 06 04 04 00 FF FF 02 00 0E 61
```

4.2.4.2 Prüfen der Korrekten Konfiguration

Um zu prüfen, ob der GPS-Empfänger korrekt konfiguriert ist, mussten folgende Befehle gesendet werden:

```
B5 62 06 07 00 00 0D 2D
```

Die erhaltenen Daten waren wie gewünscht, weshalb keine weitere Konfiguration notwendig war, siehe Abschnitt 6.1.1 auf Seite 36.

4.2.4.3 Speichern der *Current Configuration* in die *Permanent Configuration*

Da die vorgenommenen Konfigurationen nur temporär gültig sind, war es nötig, die *Current Configuration* in die *Permanent Configuration* zu übertragen, die auch ohne aktive Spannungs erhalten bleibt. Dies erfolgte mit folgendem Befehl:

```
B5 62 06 09 0C 00 00 00 00 00 FF FF FF FF 00 00 00 00 17 75
```

Sofern dieser Befehl erfolgreich ausgeführt wurde erhält man ein Acknowledge:

```
B5 62 05 01 02 00 06 09 17 40
```

4.3 Telemetry und Telecommand

Telecommands und Telemetry werden im Standard der CCSDS beschrieben und dienen der Kommunikation von Raumfahrtsystemen. Ein Raumfahrtsystem kann nach dem Package Utilization Standard *Service* mehrere *Services* bereitstellen und wird selbst über eine *ApplicationID* (APID) angesprochen. Diese *Services* können mit Telecommands aufgerufen werden und generieren, je nach *Service*, Telemetry-Daten, die zurückgesendet werden.
Um den Dienst des *Services* genauer zu spezifizieren, wird zusätzlich ein Subservice angegeben. So kann z.B. ein Service eine Seismiksensor darstellen und dessen Subservices sind *konfigurieren* und *Daten erfassen*.

Packet Header (48 Bits)							Packet Data Field (Variable)			
Packet ID				Packet Sequence Control		Packet Length	Data Field Header (Optional) (see Note 1)	Application Data	Spare	Packet Error Control (see Note 2)
Version Number (=0)	Type (=1)	Data Field Header Flag	Applica-tion Process ID	Sequence Flags	Sequence Count					
3	1	1	11	2	14					
16				16		16	Variable	Variable	Variable	16

Abbildung 4.3: Telecommand - Aufbau eines Pakets ©ECSS

Packet Header (48 Bits)							Packet Data Field (Variable)			
Packet ID				Packet Sequence Control		Packet Length	Data Field Header (Optional) (see Note 1)	Source Data	Spare (Optional)	Packet Error Control (Optional)
Version Number (=0)	Type (=0)	Data Field Header Flag	Applica-tion Process ID	Grouping Flags	Source Sequence Count					
3	1	1	11	2	14					
16				16		16	Variable	Variable	Variable	(see Note 2)

Abbildung 4.4: Telemetry - Aufbau eines Pakets ©ECSS

5 Zeitsynchronisierung

Die Zeitsynchronisierung der Sensordaten steht im Fokus dieser Bachelorarbeit. Ziel ist es die Sensordaten mit einem möglichst genauen Zeitstempel zu versehen. Im Folgenden werden die genutzte Methode und eine Genauigkeitsanalyse vorgestellt.

5.1 Methode

Die genutzte Methode wurde inspriert durch einen Application Report von Texas Instruments [Ins13].

5.1.1 Allgemeine Darstellung

In der Abbildung 5.1 auf Seite 31 ist der Grundablauf im Zeitintervall von ca. einer Sekunde dargestellt. Dort sieht man drei relevante Aktivitäten: das Eintreffen der PPS-Signale vom GPS-Empfänger, die Konvertierung der GPS-Zeit, sowie die Datenerfassungen der ADCs. Das PPS-Signal tritt in Regelmäßigen Zeinintervallen von ca. einer Sekunde auf. Nach jedem PPS-Signal beginnt die Konvertierung der GPS-Zeit, die ebenfalls vom GPS-Empfänger bereitgestellt wird. Diese Konvertierung hat eine signifikant kleinere Dauer, als in der Zeichnung dargestellt. Man sieht des Weiteren, dass in dem Zeitintervall, zweier, aufeinander folgender PPS-Signale, ca. 50 bzw. 250 Daten von den ADCs erfasst werden sollen.

Die Bereitstellung des PPS-Signals und der GPS-Zeit erfolgt jeweils über einen eigenen Interrupt. Der Interrupt des PPS-Signals ist höher priorisiert als der, der GPS-Zeit, da deren Konvertierung nicht zeitkritisch ist, der Erhalt des PPS-Signals hingegen sehr. Die

Erfassungen der ADCs laufen in einem eigenen Thread ab. Aufgrund dieser Basiskonstruktion ist eine genauere Betrachtung nötig, damit die Synchronisierung der Daten korrekte Ergebnisse liefert.

In dieser Methode ist es nötig, die CUC Coarse Time und die CUC Fine Time für den Zeitstempel getrennt voneinander zu beschaffen. Die CUC Fine Time wird auf Basis einer Zeitdifferenz zwischen dem Empfang des PPS-Signals und dem Start der jeweiligen Messung berechnet, wie in Abbildung 5.2 auf Seite 31 zu sehen ist. Die konkrete Berechnung findet sich in Abschnitt 6.6.2.3 auf Seite 44

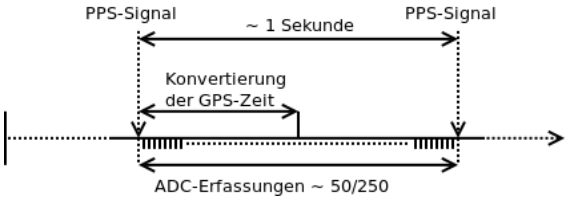

Abbildung 5.1: 1s-Intervall

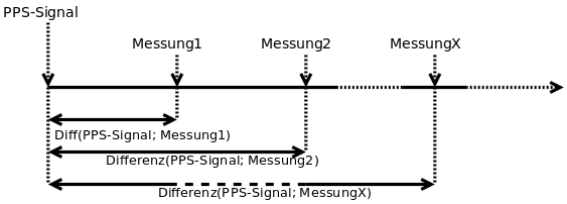

Abbildung 5.2: CUC-Fine Time

Die CUC Coarse Time wird aus der empfangenen GPS-Zeit gebildet, wie in Abschnitt 6.6.2.2 auf Seite 44 beschrieben. Die CUC Coarse Time darf immer erst nach dem PPS-Signal verfügbar sein. Dies ist allerdings kein Problem, da die Konvertierung weitaus mehr Zeit beansprucht, als die Bestimmung des Zeitpunktes des PPS-Signals.

5.1.2 Detaillierte Darstellung

In Abbildung 5.3 auf Seite 32 sieht man eine genauere Darstellung des Synchronisierungsablaufs. Zum Systemstart existiert noch keine Zeit, mit der die Messungen verknüpft werden können. Bei Empfang des ersten PPS-Signals wird eine Markierung gesetzt, die mit einer lokal verfügbaren Zeitquelle realisiert wird. Eine gleichartig gesetze Markierung wird für den Anfang einer jeden ADC-Messung verwendet. Damit ist es nun möglich die CUC Fine Time jeder ADC-Messung zu berechnen. Erst mit dem Konvertierungsende der GPS-Zeit und der damit verbundenen Verfügbarkeit der CUC Coarse Time ist es möglich, diese mit der ADC-Messung zu verknüpfen.

Beim Eintreffen des PPS-Signals wird ein Flag gesetzt, das anzeigt, dass die derzeit verfügbare CUC Coarse Time veraltet ist. Diese Information wird genutzt um bereits vor dem Konvertierungsende der GPS-Zeit eine korrekte Verknüpfung zu bilden. Es wird die veraltete CUC Coarse Time um eine Sekunde erhöht verwendet. Das zuvor erwähnte Flag wird bei Konvertierungsende der GPS-Zeit geändert, um anzuzeigen, dass die CUC Coarse Time wieder aktuell ist. Dieses Flag ist nötig, falls eine ADC-Messung beginnt, bevor die Konvertierung der GPS-Zeit abgeschlossen ist.

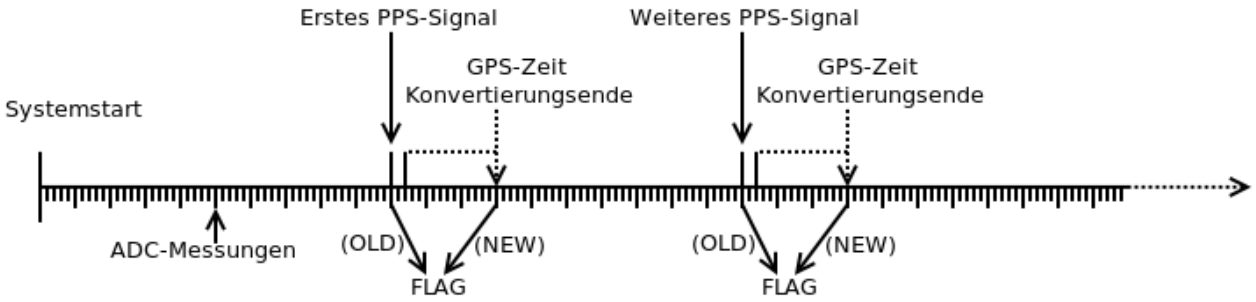

Abbildung 5.3: Detaillierter Ablauf

5.2 Genauigkeitsanalyse

Dieser Abschnitt stellt eine Analyse der erwarteten Genauigkeit der abgesetzten Zeitstempel dar. Die Analyse beginnt beim Eintreffen des PPS-Signals und endet beim Ende der Zeitstempel-Erstellung.

Die Ankunft des PPS-Signals, bzw. deren aufsteinde Flanke, stellt den Beginn der Genauigkeitsanalyse dar. Die Ankunft des Signals stellt den Interrupt-Request dar. Zu diesem Zeitpunkt besteht noch keinerlei Differenz zur GPS-Zeit. Die aufsteigende. Nun dauert es eine gewisse Zeit, bis der Interrupt aufgenommen, da der Interrupt-Request asynchron verläuft. Es ist technisch nur möglich bei jedem neuen Clock-Cycle eine Veränderung wahrzunehmen, weshalb an dieser Stelle potenziell ein Zeitverzug entsteht. Siehe Zitat:

> To ensure the NVIC detects the interrupt, the peripheral must assert the interrupt signal for at least one clock cycle, during which the NVIC detects the pulse and latches the interrupt [Mic14].

Dieser Zeitverzug liegt im Rahmen von 0 < Zeitverzug <= clock cycle-Dauer. Sofern der Interrupt akzeptiert wurde, dauert es maximal 12 Clock-Cycles[Mic14], bis der erste Befehl der Interrupt Service Routine ausgeführt wird.

Dieser erste Befehl stellt die Sicherung des Timestamps dar (aktuellen Clock-Cycle-Counter Wert). Hieraus ergibt sich die Genauigkeit für den Beginn einer Sekunde im Vergleich zur GPS-Zeit. Da die Quantifizierung von 1Hz weitaus zu ungenügend ist um exakte Messungen vorzunehmen, wird als weitere Referenz für Sekundenbruchteile die Systemuhr des Mikroprozessors genutzt.

Die Kombination beider Zeitquellen zur genaueren Bestimmung eines Zeitpunktes sorgt dafür, dass eine Ungenauigkeitsabweichung der internen Systemuhr nur für den Zeitraum von einer Sekunde auftritt, da dann die nächste exakte GPS-Zeit Zur Verfügung steht.

Nun ist hierbei zu beachten, dass eine Ungenauigkeit auftritt, welche einerseits durch die Beschränkung des verwendeten Datentyps auftritt, sowie zusätzlich durch den Quantisierungsunterschied zwischen der CUC-Fine Time und der Dauer eines CC.

Wie bereits in Abschnitten 2.3.2 und 2.2.2.1 beschrieben ergibt sich für die kleinstmögliche Quantisierung:

$$\text{Kleinstmögliche Zeitquantisierung} = \frac{1\text{s}}{2^{|Bits|}} = \frac{1\text{s}}{2^{24}} \approx 59,6046447753906\text{ns} \tag{5.1}$$

und für die Dauer eines Clock-Cycles:

$$\text{CC-Dauer} = \frac{1\text{s}}{\text{Taktfrequenz}} = \frac{1\text{s}}{168\text{MHz}} = 5,9523\overline{809523}\text{ ns} \tag{5.2}$$

Die Ungenauigkeit, die sich hieraus ergibt liegt im also im Bereich von 0 Nanosekunden bis ca. 59,6046447753906 Nanosekunden, je nachdem, wie viele Clock-Cycles bereits gezählt wurden.

5.3 Verwandte Arbeit

In diesem Abschnitt wird kurz die simple Methode beschrieben, die die tatsächliche Implementierung inspiriert hat [Ins13]. In der genannten Quelle soll eine DP83640 Zeitreferenz synchronisiert werden. Hierzu wird das PPS-Signal eines GPS-Empfängers genutzt um die Zeitsynchronisierung vorzunehmen.

Zunächst wird bei Autreten der PPS-Flanke ein Timestamp abgesetzt. Daraufhin wird die GPS-Zeit vom GPS-Empfänger gelesen. Im Anschluss wird ein Zeitversatz zur GPS-Zeit gebildet. Dieser Vorgang soll in regelmäßigen Intervallen wiederholt werden, um die Präzision zu gewährleisten.

6 Implementierung

Dieser Abschnitt stellt konkretere Abläufe und zum Teil auch Source-Code der Implementierung vor.

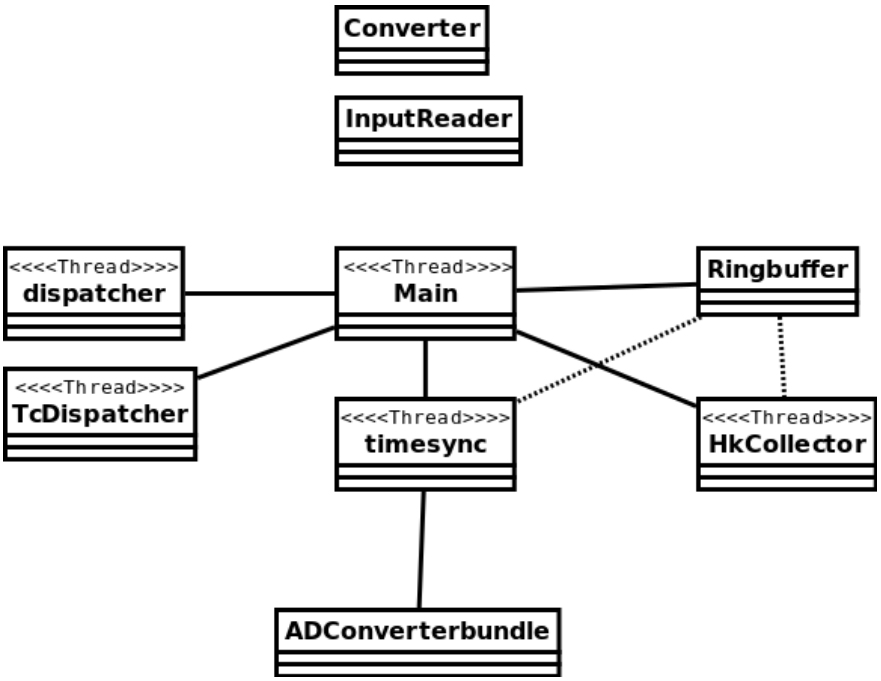

Abbildung 6.1: Programmaufbau

In Abbildung 6.1 auf Seite 35 sieht man die Beteiligten Hauptkomponenten der Implementierung. Der Main-Thread beinhaltet die Konfiguration der Hardware, die Behandlung der Interrupts, sowie das Starten der anderen benötigten Threads. Der Main-Thread hält indes auch einen Ringbuffer, der für die Aufnahme der Messdaten der Analog-Digital-Wandler gebraucht wird. Dieser wird dem *HkCollector* und *timesync* als Referenz bereitgestellt.
Des Weiteren gibt es Threads, die die Aufnahme und den Versand von Telemetry und Telecommands abwickeln. Diese sind der *TcDispatcher*, der *dispatcher* und der *HkCollector*.

Es gibt einen Thread *timesync*, der die Erfassung der Analog-Digital-Wandler zeitlich koordiniert und dazu ein *ADConverterBundle* verwendet. Zusätzlich gibt es zwei Klassen *Converter* und *InputReader*, die die Aufnahme und Konvertierung der GPS-Daten Vornehmen.

6.1 Hardware-Konfiguration

6.1.1 GPS-Empfänger

Die Konfiguration des GPS-Empfängers wurde vor dem Anschließen an den das System mit einem Laptop in einem Terminal vorgenommen.

Die hierfür verwendeten Befehle wurden bereits in Abschnitt 4.2.4 auf Seite 27 erläutert.

Der GPS-Empfänger muss bei erstmaliger Benutzung mittels Cold-Start aktiviert. Ähnliches gilt bei einer Nicht-Verwendung von längerer Zeit als einigen Tagen, oder einem signifikanten Standortwechsel.

Je nachdem, welche Daten der GPS-Empfänger in seinem aktuellen Zustand besitzt ist einer der Befehle am geeignetsten um ihn in Bereitschaft zu versetzen. Bei einer noch nie gestarteten oder lange inaktiven Komponente ist ein Cold-Start zu empfehlen, weil die vollständige Auswertung der Almanac- und Ephemeris-Daten erfolgt, die bei einem Hot-Start als bereits erledigt vorrausgesetzt wird. Nach einer kurzen Wartezeit von ca. 40 Sekunden bei klarem Himmel ist das Gerät einsatzbereit.

Der GPS-Empfänger liefert nach Aktivierung im Werkszustand viele verschieden Datensätze im ASCII-Format. Da für die Ziele nur der GPZDA-Datensatz wichtig ist, wird mittels NMEA-Befehlen das Senden aller anderen Datensätze deaktiviert und das Senden des GPZDA-Datensatzes explizit aktiviert.

Da diese Konfiguration nicht persistent ist wird sie explizit in den permanenten Speicher übertragen, so dass nach einem kurzzeitigen Ausschalten des Gerätes keine erneute Konfiguration nötig wird.

Die Konfiguration des PPS-Signals ist Standardgemäß auf eine Frequenz von 1 Hz eingestellt mit einer Flankendauer von 100ms. Diese Konfiguration wurde mit einem Befehl überprüft und schein geeignet zu sein.

6.1.2 Seismik-Sensor / Analoge Signalquelle

Da der eigentliche Seismik-Sensor derzeit nicht angeschlossen werden kann, werden zur Simulation verschiedene Batterien und ein Frequenzgenerator an die GPIO-Pins (General Purpose Input Output) angeschlossen.

6.2 Software-Konfiguration

In diesem Abschnitt wird die Software-Konfiguration der verschiedenen Hardwarekomponenten vorgestellt. Hierunter fällt hauptsächlich die Aktivierung bestimmter Funktionalitäten und die Festlegung von verwendeten GPIO-Pins.

6.2.1 GPS-Empfänger

Die Software-Konfiguration des GPS-Empfängers wird in der Datei *init.cpp* vorgenommen. Es werden hierfür ein UART-Port für die Kommunikation konfiguriert, zwei Software-Relais zur Spannungsversorgung gesetzt und ein Interrupt-Vector eingestellt und aktiviert. Zusätzlich wird für die Nutzung des PPS-Signals ein GPIO gesetzt, sowie ein weiterer Interrupt-Vector eingestellt und Konfiguriert. Der eingestellte Interrupt-Vector des PPS-Signals hat eine höhere Priorität als die der RX-Leitung. Im Folgenden ist ein Code-Auszug der Konfiguration zu sehen:

```
gps_tx::connect(Usart6::Tx);
gps_rx::connect(Usart6::Rx, Gpio::InputType::PullUp);
gpsUart.initialize<DefaultSystemClock, 9600>(10);

gps_vcc::setOutput(xpcc::stm32::Gpio::OutputType::PushPull);
gps_vcc_red::setOutput(xpcc::stm32::Gpio::OutputType::PushPull);

gps_rx::setInputTrigger(Gpio::InputTrigger::RisingEdge);
gps_rx::enableExternalInterrupt();
gps_rx::enableExternalInterruptVector(11);

gps_pps_input::setInputTrigger(Gpio::InputTrigger::RisingEdge);
gps_pps_input::enableExternalInterrupt();
gps_pps_input::enableExternalInterruptVector(12);
```

6.2.2 Mikroprozessor

Wie bereits in Abschnitt 2.2.2.1 dargestellt wurde, soll die präziseste Zeitquelle mit der geringsten Quantisierung gewählt werden, um die Sekundenbruchteile zu berechnen. Da der lesende Zugriff auf den Clock-Cycle-Counter möglich ist, wird dieser verwendet.

Bei einem STM32F427ZGT6-Mikroprozessor ist die Aktivierung der Datawatch-Tracetable erforderlich [NAa]. Mit der Aktivierung ist es möglich auf den Clock-Cycle-Counter zuzugreifen und nutzbare Daten zu erhalten. Diese Funktion ist eigentlich für das Debugging gedacht, wird jedoch benutzt um exakter den Zeitpunkt des PPS-Signals zu bestimmen.

Die Aktivierung der Datawatch-Tracetable erfolgt durch das Setzen des Registers DEMCR (Debug Exception and Monitor Control Register) auf den Wert 0x01000000. Im Anschluss wird der Clock-Cycle-Counter auf 0 gesetzt. Hierfür muss der Wert in dem Register DWT_CYCCNT auf 0 gesetzt werden. Als letzter Schritt wird die Zählung aktiviert. Dies geschieht durch das Setzen jeden Bits des DWT_Control Registers auf den Wert 1. Im Folgenden der entsprechende Codeabschnitt, welcher sich in der Datei *init.cpp* befindet:

```
//enable the use DWT
*DEMCR = *DEMCR | 0x01000000;
//reset cycle counter
*DWT_CYCCNT = 0;
//enable cycle counter
*DWT_CONTROL = *DWT_CONTROL | 1;
```

Des Weiteren wird die Frequenz des Prozessors auf 168 MHz festgelegt.

6.2.3 Analog-Digital-Konverter

Die Konfiguration der Analog-Digital-Wandler des Mikroprozessors erfordert lediglich die Zuordnung zu General Purpose Input Output Pins (GPIO-Pins), sowie einer Angabe über den gewünschten Kanal des Analog-Digital-Wandlers. Diese Konfiguration erfolgt fest in der Klasse *ADConverterBundle* (siehe Abschnitt 6.5 auf Seite 41). Eine Änderung erfordert den Eingriff in den Quell-Code. Die verwendeten GPIO-Pins werden mit Hilfe von *XPCC* gesetzt, anhand einer automatisch Zuordnung für die gewählte Plattform.

6.3 Eigene Datentypen

Für die interne Arbeitsweise der Software werden eigene Datentypen verwendet, die in Folgendem kurz erläutert werden sollen.

6.3.1 GPSTimeObject

Es existiert eine Klasse GPSTimeObject, die erstellt wurde um auf einfache Weise die Konvertierung der, in ASCII-Zeichen gelieferten, GPS-Zeit, vorzunehmen. Der Datentyp findet sich in der Datei *timeTypes.h* und sieht aus wie folgt:

```
class GPSTimeObject
{
```

```
  private:
    uint16_t year = 0;
    uint8_t month = 0;
    uint8_t day = 0;
    uint8_t hour = 0;
    uint8_t minute = 0;
    uint8_t second = 0;
    // Es folgen weitere Methoden für den Zugriff
}
```

Die Klasse beinhaltet Einträge für die einzelnen Teile eines regulären Datums, wie Jahr, Monat oder Tag. Zu beachten ist hierbei, dass keine Sekundenbruchteile in diesem Datentyp existieren.

6.3.2 PayloadEntry

Für die Einbettung der Daten in das Housekeeping Frame wird ein PayloadEntry-Format verwendet, welches in Abbildung 6.2 auf Seite 40 zu sehen ist. Der Datentyp findet sich in der Datei *PayloadEntry.h*.

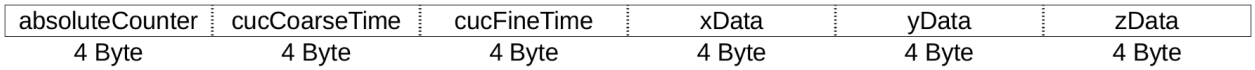

Abbildung 6.2: PayloadEntry-Format

Das PayloadEntry-Format besteht aus einem *absoluteCounter*, einer *cucCoarseTime*, einer *cucFineTime*, sowie *xData*, *yData* und *zData*. Der *absoluteCounter* ist ein allgemeiner Zähler, mit dem man die absolute Ordnung der Messdaten eines Messvorgangs herstellen kann. Die *cucCoarseTime* und die *cucFineTime* bilden eine vollständige CUC-Zeit, bestehend aus vier Oktetten Coarse Time und drei Oktetten Fine Time. Die Einträge *xData*, *yData* und *zData* beinhalten für jede Achse einen, von den ADCs bereitgestellten, Spannungswert. Dieses PayloadEntry-Format wird als *Payload* in ein TM-Paket eingebunden.

6.4 Zeitsynchronisierung

Die Zeitsynchronisierung wird mit Hilfe von verschieden priorisierten Interrupts vorgenommen, die in der Datei *init.cpp* zu finden sind und im Main-Thread ausgeführt werden. Es existieren global zwei uint32_t Variablen *actualCUCCoarseTime* und *secondStartClockCycleStamp* die einerseits die aktuelle CUC Coarse Time speichern und andererseits den aktuellen Stand der Clock-Cycles zum Start der Sekunde markieren. Der hochpriorisierte interrupt empfängt das PPS-Signal und tut nichts weiteres als den aktuellen Clock-Cycle-Stand zu speichern. Der niedrigpriorisierte Interrupt übergibt die im UART enthaltenen Zeichen an die Klasse *Timesync*. Diese Klasse implementiert den in Abschnitt 6.6.1 enthaltenen Pseudo-Code. Zusätzlich wird von *Timesync* die Konvertierung zur CUC Coarse Time eingeleitet, die in Abschnitt 6.6.2.2 zu finden ist.

6.5 Datenerfassung

Die Datenerfassung geschieht mit der Klasse ADConverterBundle, welche die verwendeten Analog-Digital-Wandler konfiguriert, und diese verwendet, um deren Daten verknüpt mit der CUC-Zeit in einem Ringbuffer zu speichern. Die Klasse selbst leitet sich von einer *Thread-Klasse* ab.

Der Vorgang läuft ab wie folgt:

1. Speichern des aktuellen Clock-Cycle-Counter Werts
2. Start aller ADC-Samplings
3. Warten auf die Beendigung des Samplings
4. Konvertierung der erfassten Daten
5. Konvertierung der CUC Fine Time
6. Abgreifen der aktuellen CUC Coarse Time
7. Einbettung in einen Payload-Entry
8. Sicherung des erstellten Payload-Entrys in den Ringbuffer

9. Erhöhung des absoluten Zählers der Messungen

Initial wird der ADConverterBundle-Thread mit der Frequenz eines passiven Experiments (50 Hz) gestartet. Daraus ergibt sich eine Wartezeit von 20ms bis zur jeweils nächsten Messung. Es ist möglich diese Frequenz über einen Methodenaufruf auf ein passives Experiment abzuändern. Somit wird die Frequenz auf 250 Hz gesetzt und die Wartezeit zwischen jeder Messung beträgt nur noch 4ms. Auch ein passives Experiment kann jederzeit wieder durch einen Methodenaufruf initiiert werden.

6.6 Hilfsklassen

Die Hilfsfunktionen dienen der Bereitstellung der CUC-Zeit. Sie wird im Grundsatz mit zwei Klassen durchgeführt, die in Abbildung 6.3 auf Seite 6.3 zu sehen sind. Alle anderen Klassen benutzen diese, um die CUC-Zeit zu erhalten.

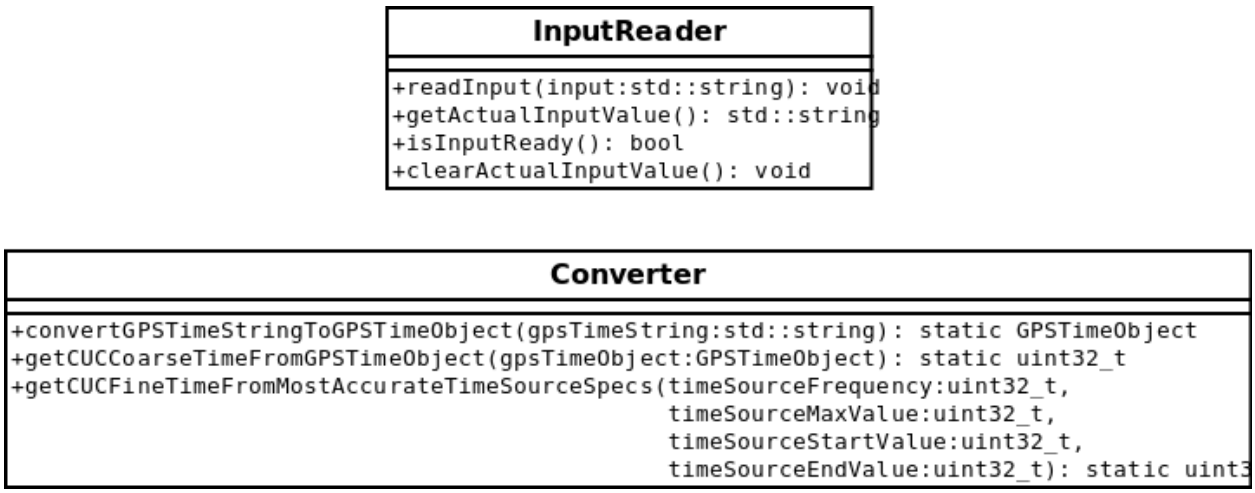

Abbildung 6.3: Klassen zur Bereitstellung der CUC-Zeit

6.6.1 InputReader

Der *InputReader* liest ein oder mehrere Zeichen ein und bildet durch Konkatenation intern eine Zeichenkette (Datentyp ist std::string). Sobald er ein *endSymbol* ('\n') einliest, wird intern die boolsche Variable *inputReady* auf *true* gesetzt um zu symbolisieren, dass eine Zeichenkette in der erwarteten Formatierung (NMEA Message) eingelesen wurde. Diese

Zeichenkette kann nun von anderen Funktionen angefordert werden, um sie auszuwerten.

Der *InputReader* muss verwendet werden, wie in folgendem Pseudocode zu sehen ist:

```
while(1){
inputReader.readInput(inputString);

if(inputReader.isInputReady()){
  outputString = inputReader.getActualInputValue();
  inputReader.clearActualInputValue();
}
}
```

Zuerst wird dem InputReader eine (zumeist unvollständige) Zeichenkette übergeben. Anschließend wird der Status auf Verfügbarkeit einer vollständigen Zeichenkette erfragt. Sollte der Status *true* sein, so kann die Zeichenkette angefordert werden, verarbeitet und/oder gespeichert. Sollte der Status *false* sein, so sind weitere Zeichenkettenübergaben nötig. Sofern man eine vollständige Zeichenkette im NMEA-Message-Format erhalten hat, muss man dem InputReader mitteilen, dass diese nicht mehr benötigt wird. Dies geschieht mit *clearActualInputValue()*. Im Anschluss kann die Bildung einer neuen Zeichenkette beginnen.

6.6.2 Converter

Der *Converter* dient dazu verschiedene Konvertierungen zu bewerkstelligen, welche CUC-Zeit und GPS-Zeit betreffen. Er ermöglicht die Konvertierung eines einer Zeichenkette im NMEA-Message-Format in ein GPSTimeObject, eines GPSTimeObjects in uint32_t (CUC Coarse Time) und einer Zeitdifferenz zweier Zeitstempel in uint32_t (CUC Fine Time).

6.6.2.1 convertGPSTimeString

Diese Funktion konvertiert eine Zeichenkette im NMEA-Message-Format in ein GPSTimeObject. Falls eine valide GPZDA-Standard-Message mit ausreichend Daten übergeben

wird, extrahiert sie die Daten und gibt sie als GPSTimeObject zurück. Falls keine valide GPZDA-Standard-Message übergeben wurde, so wird ein Default GPSTimeObject zurückgegeben, dessen einzelne Variablen jeweils Nullwerte sind. Es wird hier also eine Art Filterung durchgeführt.

6.6.2.2 getCUCCoarseTimeFromGPSTimeObject

Diese Funktion konvertiert ein GPSTimeObject in eine CUC Coarse Time. Hierfür wurde eine Variation einer libcobc-Funktion genutzt. Die Funktion berechnet die absolute Anzahl an Sekunden ab der, in diesem Projekt festgelegten Epoche (01.01.2000). Hierbei werden keine Leap Seconds beachtet, da sie in der GPS-Zeit nicht existieren.

6.6.2.3 getCUCFineTimeFromMostAccurateTimeSourceSpecs

Diese Funktion berechnet anhand der übergebenen Frequenz, deren Maximalwert und zweier Zeitstempel einer Zeitquelle, Sekundenbruchteile in 24-Bit genauer Quantisierung. Hieraus ergeben sich Ungenauigkeiten im Bezug zu der Tatsächlichen Zeit, im Bereich von weniger als einer Nanosekunde wie in Abschnitt 5.2 auf Seite 32 bereits erläutert.

Um die Sekundenbruchteile zu ermitteln wird zunächst die Differenz zweier Clock-Cycle-Counter gebildet. Dass der Clock-Cycle-Counter nach seiner Begrenzung von 2^{24} wieder von 0 beginnt wurde hierbei beachtet.

Die Formel hierfür ist:

```
clockCycleCounterDiff * ()(1 << 24) / (double) clockCycleFrequency)
```

6.7 Kommandoempfang und Datenversand

Die Kommunikation im System findet über TC und TM statt. Dazu werden Telecommands auf einem Client präpariert und über eine Serielle Verbindung an das den On-Board-Computer weitergeleitet. Dazu wird das TC mit Hilfe der Klasse HDLC in ein Frame umgewandelt. Der On-Board-Computer stellt das TC wieder her und leitet das

Telecommand an das *telecommandDistribution* Topic weiter. Auf diesem Topic horcht der *TelecommandDispatcher*, der die Telecommands entgegennimmt und an die *Applications* weiterleitet. In diesem konkreten Fall wird es an die Klasse *HKDownlinker* weitergeleitet. Diese *Appplication* hat einen *Service* der durch die Klasse *HKDownlinkerService* implementiert ist. Die von dem *Service* generierten Telemetrie-Pakete werden an das *telemetryReporting*-Topic weitergeleitet und gelangen dadurch zum *Downlinker*. Dieser serialisiert das TM und gibt es ebenso über eine serielle Schnittstelle an den Client weiter.

Zwei konkrete Befehle die ausgeführt werden sind *startPassiveExperiment()* und *startActiveExperiment*. Diese beiden Befehle werden über den *HKDownlinkerService* ausgeführt. Hierzu verwendet er einen Pointer auf die Timesync-Instanz, die wiederum Zugriff auf die Frequenzmanipulation des *ADConverterBundles* hat. So wird mittels Telecommand ein *Aktives Experiment* oder ein *Passives Experiment* initiiert.

7 Testen

Hier aufgeführt sind die verschiedenen Testmittel um die Korrektheit der Software zu überprüfen. Beginnend beim Quellcode, verschiedenen Praxistests, sowie die Benennug von Problemen, die die Hardware mit sich führt.

7.1 Quellcode

Im Rahmen der Implementierung wurden Unittests, sowie Integrationstests durchgeführt. Verwendet wurde hierfür die Cpputest Testharness. Getestet wurden somit primär der *InputReader*, der *Converter* sowie der, auf den Datentyp *PayloadEntry* beschränkte, *Ringbuffer*. Die konkreten Tests befinden sich auf dem beiliegenden Datenträger.

7.2 Praxistest

Im Folgenden werden die Praxistests präsentiert. Hierfür wurden mit Hilfe eines Sweep/-Function Generators verschiedene Signale Simuliert und Telecommands verschickt, die die Frequenz der Datenerfassung manipulieren sollen. Die Daten wurden von der Software erfasst und werden in den folgenden Grafiken ausgewertet. Leider konnte zum Zeitpunkt der Messung keine GPS-Zeit und kein PPS-Signal empfangen werden, da die Tests im Gebäude stattfanden.

7.2.1 Testvorgang

Die Tests wurden vor allem eine Debug-Ausgabe des *ADConverterBundle* benutzt um die Änderung der Signale zu betrachten. Zeitgleich wurden die empfangenen Daten in CSV-Form in einem Terminal abgegriffen. Der Versand der Daten als CSV zum Auswerten war leider nur bedingt möglich, da die Serialisierung sehr inperformant gelöst wurde und eine enorme Wartezeit bis zum Erhalt der Daten verstrich. Hier ist es auf jeden Fall nötig zu optimieren, um die Daten tatsächlich in Echtzeit nutzen zu können.

7.2.1.1 Passive Experimente (50 Hz)

Diese Experimente stellen der Verlauf verschiedener Signalformen dar. Die verschiedenen Signalformen waren Sinus-, Rechteck- und Sägezahnschwingungen auf Basis von 1,002 Hz und 10,12 Hz vorgenommen.

7.2.1.1.1 Durchlauf 1 - 1,002 Hz Die Abbildung 7.1 zeigt den ersten Durchlauf von 50 Samples, die erfasst wurden. Man sieht dort alle drei Signalformen, die sehr stark den optimalen Kurven gleichen.

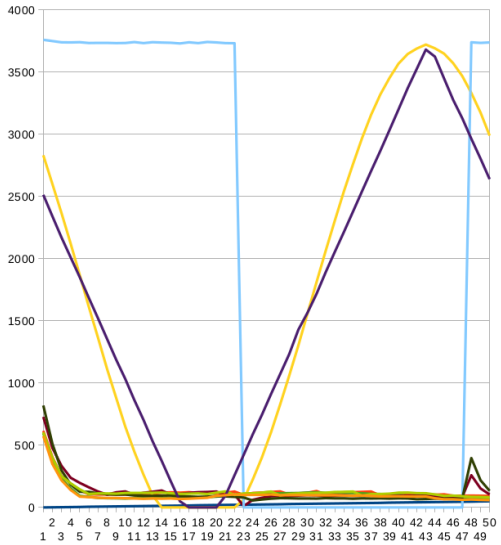

Abbildung 7.1: Durchlauf 1

7.2.1.1.2 Durchlauf 2 - 10,12 Hz Die Abbildung 7.1 zeigt den zweiten Durchlauf von 50 Samples, die erfasst wurden. Man sieht dort alle drei Signalformen.

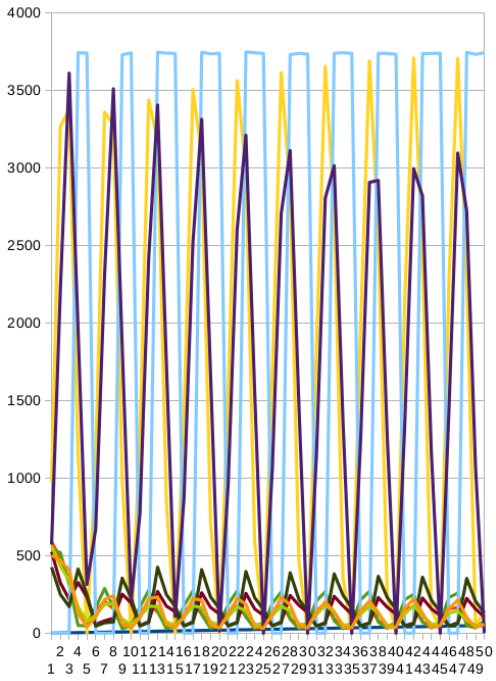

Abbildung 7.2: Durchlauf 2

7.2.1.2 Initiierung eines Aktiven/Passiven Experiments

Es wurde versucht die Initiierung eines Aktiven/Passiven Experiments durchzuführen. Dies misslang, da die Reihenfolge in der die Threads der Software ausgeführt wurden, Einfluss auf die Funktionalität haben.
So kann bei einem zuerst startenden HKCollector kein Telecommand ausgeführt werden. Bei einem zuletzt startenden HKCollector konnten Kommandos ausgeführt werden, jedoch keine Telemetrie-Daten verschickt werden. An dieser Stelle muss geprüft werden, wie die Abhängigkeiten der Threads untereinander sind um beide Funktionalitäten gleichzeitig nutzen zu können.

7.2.2 Probleme im Praxistest

Der GPS-Empfänger ist auf den Empfang von Satellitendaten angewiesen, die leider Wetterabhängig sind und, vor allem in Gebäuden zu großen Problemen führen. So ist es mehrfach vorgekommen, dass komplette Abende lang, aufgrund von Wolkendecken und Regen, keine brauchbaren Daten empfangen werden konnten. Dies gilt sowohl für die Verfügbarkeit der GPS-Zeit, als auch der Verfügbarkeit des PPS-Interrupts, der bei vielen Versuchen frühestens nach ca. einer Minute Betriebsdauer erstmalig aufgetaucht ist.

7.2.3 Sonstige Tests

Bei der Entwicklung der einzelnen Komponenten war die Verfügbarkeit der eGPS-Daten höher als bei der Durchführung der beiden Testläufe, wodurch das korrekte Verhalten festgestellt werden konnte. So arbeiten der InputReader, der Converter und der Ringbuffer wie erwartet. Auch die Bereitstellung der CUC Coarse und CUC Fine Time erfolgte wie erwartet.

8 Zusammenfassung und Ausblick

Dieser Abschnitt enthält eine Zusammenfassung der Ergebnisse, sowie einen Ausblick auf zukünftige Tätigkeiten, sowie Fragestellungen.

8.1 Zusammenfassung

Die Zeitsynchronisierung kann mit der Software erfolgreich durchgeführt werden. Die praktisch verfügbare Genauigkeit konnte leider nicht genügend geprüft werden. Die Einzelkomponenten arbeiten zum größten Teil sehr zuverlässig, bis auf einige inperformante Vorgänge. Der gleichzeitige Empfang von Telecommands und das Senden von Telemetrie-Daten konnte nicht erreicht werden, einzeln jedoch funktioniert dies bereits. Insgesamt können die gesetzten Ziele der Software-Implementierung zum Großteil als erfüllt betrachtet werden.

8.2 Ausblick

8.2.1 Ausfall der Zeitquelle

Eine Frage, die aufgetaucht ist, ist der Ausfall der genutzten Zeitquelle. Im konkreten Bezug zu dieser Arbeit stellt sich die Frage, wie das System reagieren soll, wenn die Zeitquelle keine Daten mehr liefert.

8.2.2 Defekt der Zeitquelle

Falls die Zeitquelle defekt ist, wie ist damit umzugehen? Wie kann dies festgestellt werden?

8.2.3 Substituierung der Hardware-Komponenten

Wie bereits im Abschnitt 3.2.4 beschrieben, ist für die Zukunft geplant andere Hardware-Komponenten zu verwenden. Hierunter fällt der Austausch der Analog-Digital-Wandler und der Analogen Datenquelle (Seismik-Sensor/Batterie/Frequenzgenerator).

A Anhang

Auf dem Beiliegenden Datenträger ist der Quellcode der Software enthalten. Er teilt sich auf in den Source-Code, der für die konkrete Implementierung verwendet wurde und den Source-Code, indem mittels TestDrivenDevelopment einige Einzelkomponenten entwickelt wurden. Eine Library, die in dieser Bachelorarbeit verwendet wurde, darf aus rechtlichen Gründen nicht zur Verfügung gestellt werden.

Literaturverzeichnis

[AG08] AG u-blox: *NMEA, UBX Protocol Specification*. 08 2008

[Com15] Comer, Douglas: *Operating system design : the Xinu approach*. Boca Raton, FL : CRC Press, Taylor & Francis Group, 2015. – ISBN 978–1–4987–1244–6

[ele11] electronic lennartz: *LE-xD Seismometer Family Document Number: 990-0003*. 02 2011

[IncNA] Inc., GlobalTop T.: *What differentiate between Cold Start, Warm Start and Hot Start?* http://www.gtop-tech.com/en/faq/4.9.-What-differentiate-between-Cold-Start-Warm-Start-and-Hot-Start/GPS_Module_Design-4_9.html. Version: NA. – Aufgerufen am: 02.12.2015 - 17:33

[Ins13] Instruments, Texas: *AN-2006 Synchronizing a DP83640 PTP Master to a GPS Receiver*. 04 2013

[KB06] Ken Behrendt, Ken F.: *The Perfect Time: An examination of Time-Synchronization Techniques*. 2006

[Mic12] Microelectronics, ST: *Reference Manual - STM32F40x, STM32F41x, STM32F42x, STM32F43x advanced ARM-based 32-bit MCUs*. 2012

[Mic14] Microelectronics, ST: *STM32F3 and STM32F4 Series Cortex-M4 programming manual*. 2014

[Mur] Murphy, Dr. M.: *Operating Systems*. https://www.youtube.com/watch?v=2Z0yIguC5eU&list=PL56B156F5D73BBD77

[NAa] NA: *How to count cycles on ARM Cortex M*. http://embeddedb.blogspot.de/2013/10/how-to-count-cycles-on-arm-cortex-m.html

[NAb] NA: *ROBEX - Demo-Missionen*. http://www.robex-allianz.de/ueber-robex/demo-missionen/

[NAc] NA: *ROBEX - Ziele*. http://www.robex-allianz.de/ueber-robex/ziele/

[Pac96] Packard, Hewlett: *AN-1272 GPS and Precision Timing Applications*. 1996

[SI07] Spectrum Instruments, INC.: *Summary of Time and Frequency Systems*. 2007

[SiRNA] SiRF: *TTFF (Hot start / Warm start / Cold start)*. http://faq.holux.com/images/f/fd/What_is_GPS_cold_start.pdf. Version: NA. – Aufgerufen am: 02.12.2015 - 17:37

[Sta12] Stallings, William: *Operating systems : internals and design principles*. Boston : Prentice Hall, 2012. – ISBN 978–0–13–230998–1

[Tan08] Tanenbaum, Andrew: *Modern operating systems*. Upper Saddle River, N.J : Pearson Prentice Hall, 2008. – ISBN 978–0136006633

[Tan11] Tanenbaum, Andrew: *Computer networks*. Boston : Pearson Prentice Hall, 2011. – ISBN 0132126958

[The10] The Consulative Committee for Space Data Systems (CCSDS): Time Code Formats. In: *Blue Book* (2010), Nr. 4. http://www.ccsds.org/documents/pdf/CCSDS-101.0-B-4.pdf. – Aufgerufen am: 02.12.2015 - 16:20

[TSD15] Time Service Dept., U.S. Naval O.: *Leap seconds*. http://tycho.usno.navy.mil/leapsec.html. Version: 2015. – Aufgerufen am: 02.12.2015 - 16:18

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Ort, Datum Unterschrift